Brief Announcement: Model Checking Transactional Memory with Spin

John O'Leary
Intel
john.w.oleary@intel.com

Bratin Saha
Intel
bratin.saha@intel.com

Mark R. Tuttle
Intel
tuttle@acm.org

Abstract: We used the Spin model checker to show that Intel's implementation of software transactional memory is correct, and built a preprocessor to accelerate the performance of Spin on parameterized models of shared-memory protocols.

Categories and Subject Descriptors: B.6.3 [**Logic Design**]: Design Aids—*Verification*
General Terms: Algorithms, Design, Verification
Keywords: Transactional memory, Model checking, Spin

Transactional memory is a programming abstraction for multi-threaded programming that provides thread synchronization without requiring the explicit use of locks. Locks are notoriously difficult to use correctly: They can lead to deadlock, priority inversion, and subtle errors, and they can impede compositionality. To the programmer, transactional memory has a very simple interface: Every block of code labeled "atomic" is executed as an "atomic transaction" without any interferences from other threads. To the implementer, transactional memory is an introduction to the fascinating and subtle world of shared-memory protocols.

Our goal is to learn to model check shared-memory algorithms in general, and transactional memory in particular. We take the Intel McRT STM [1] implementation of transactional memory as our example, which is itself an interesting, innovative, and subtle shared-memory protocol. We choose Spin [4] as our model checker because it is engineered for high performance, it is famous for its partial order reduction algorithm which has the effect of reducing the branching factor of the transition graph, and its input language, Promela, is a natural language for describing the sequential natural of each thread in a concurrent program. Spin assumes processes communicate through message channels, but it does have global variables, and what is shared memory but a collection of global variables? The use of global variables, however, decimates the performance of the partial order reduction algorithm, which was the whole reason for choosing Spin in the first place.

This paper makes two contributions:

1. **The validation of an industrial implementation of transactional memory.** We used the Spin model checker [4] to prove the following: McRT STM guarantees the serializability of every execution of every purely-transactional program consisting of two threads each running one transaction consisting of three reads or writes. (By purely-transactional, we mean that every read and write occurs within a transaction.) Our model of McRT STM is very close to the C++ code used in the actual McRT STM implementation, in contrast to other applications of model checking to transactional memory that use very abstract models of the implementations [2, 3]. Our model of program execution is very efficient, in that checking the correctness of just a single program described above takes almost a minute (45 seconds), but we can check the correctness of all 14,400 such programs in just over an hour (1:06).
2. **A tool to optimize models of shared-memory algorithms in Spin.** We have gained a deeper understanding of how to model shared-memory algorithms in Spin, and we have built a Spin preprocessor `spp` that captures some of this insight. The preprocessor rewrites our Spin code in a way that lets the Spin compiler apply its optimizations more effectively and dramatically improves the performance of the resulting model checking task. The preprocessor allows us to write the fully-parametrized models any good computer scientist would be inclined to write — with parameters for the number of threads, length of transactions, number of memory locations, etc. — but to generate for any particular instantiation of these parameters a model that runs much faster under Spin than the parametrized model itself.

[1] A. Adl-Tabatabai et al. Compiler and runtime support for efficient software transactional memory. In *Proceedings of the ACM Conference on Programming Language Design and Implementation*, pages 26–37, June 2006.

[2] R. Alur, K. McMillan, and D. Peled. Model-checking of correctness conditions for concurrent objects. In *Proceedings of the IEEE Symposium on Logic in Computer Science*, pages 219–228, July 1996.

[3] A. Cohen, J. O'Leary, A. Pnueli, M. R. Tuttle, and L. Zuck. Verifying correctness of transactional memories. In M. Sheeran and J. Baumgartner, editors, *Proceedings of the Symposium on Formal Methods in Computer Aided Design*, pages 37–44, November 2007.

[4] G. Holzman. *The Spin Model Checker: Primer and Reference Manual.* Addison-Wesley, 2004.